Self-Speculative Decoding for On-device MoE Acceleration

Peirong Zheng
Department of Computing, The Hong Kong Polytechnic University
Hong Kong, China
peirong.zheng@connect.polyu.hk

Wenchao Xu*
Division of Integrative Systems and Design, The Hong Kong University of Science and Technology
Hong Kong, China
wenchaoxu@ust.hk

Haozhao Wang
School of Computer Science and Technology, Huazhong University of Science and Technology
Wuhan, China
hz_wang@hust.edu.cn

Abstract

The sparse mixture-of-experts (MoE) architecture is a promising backbone of foundation models for a wide range of applications in edge. However, deploying them locally presents a significant challenge to memory-constrained GPUs. Previous techniques utilize CPUs for expert offloading, which suffer from inaccurate expert prefetching and on-demand loading latency. To address these challenges, we propose self-speculative MoE (*SS-MoE*), an algorithm-system co-design framework that facilitates inference under limited GPU memory. Our insight is that only a subset of routed experts, i.e., draft model, can still tackle easy tasks and generate draft tokens. Second, we deem GPU memory as the experts cache, and on-demand update it to mitigate IO overhead. Draft tokens from fewer routed experts are generated quickly, and these experts are then routed for verification. Additionally, we design a confidence-based policy to adaptively accept or verify draft tokens, which selectively decreases or increases the number of verification tokens of speculative decoding and achieves acceleration. Notably, under conservative verification, our approach preserves model accuracy and surpasses the decoding speed of the 4-bit quantized counterpart model. Under adaptive verification, our method significantly enhances decoding speed by 3.72x over state-of-the-art methods while maintaining nearly lossless accuracy.

CCS Concepts

• **Computing methodologies** → **Natural language generation**.

Keywords

Large Language Model; Speculative Decoding; Mixture of Experts

ACM Reference Format:
Peirong Zheng, Wenchao Xu, and Haozhao Wang. 2026. Self-Speculative Decoding for On-device MoE Acceleration. In *Proceedings of the ACM Web Conference 2026 (WWW '26), April 13–17, 2026, Dubai, United Arab Emirates.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3774904.3792218

1 Introduction

Mixture-of-experts (MoE) models have recently gained significant attention, emerging as a dominant architecture due to their computational efficiency in many large language models (LLMs) [1]. By sparsely activating only a fraction of their parameters for each input, MoE models offer substantial opportunities for optimization. This characteristic makes them particularly well-suited for on-device deployment in edge computing scenarios, where local inference can ensure user privacy and provide low-latency responses, enabling edge agent applications [2], such as a personal assistant. However, the substantial memory footprint required to store all experts presents a formidable challenge for consumer-grade GPUs with limited memory.

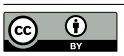
Consequently, prior research has largely focused on memory optimization strategies. First, prefetching-based methods attempt to load necessary experts from slower memory (i.e., CPU RAM) into GPU memory before they are needed [3, 4]. However, its efficacy is often hindered by the difficulty of accurately predicting all routed experts through computing routing weights using the last layer's input, which leads to on-demand loading stalls and high latency. Instead of relying on the MoE routers, [5] introduces an auxiliary trained predictor model, but it adds significant training overhead and deployment complexity, making such solutions impractical.

Another computation offloading approach utilizes CPU to compute a portion of the experts, thereby avoiding the overhead of expert movement. While this reduces memory transfer latency, the slower CPU processing speed often becomes the new performance bottleneck [6]. Approaches like SwapMoE [7] attempt to mitigate this by load balancing and experts pruning, but this comes at the cost of complicated configuration for heterogeneous hardware and degraded model accuracy.

From algorithm perspective, speculative decoding [8], a prominent technique for accelerating auto-regressive decoding, has been widely regarded as ineffective for MoE models. For instance, while Eagle [9] achieves up to a 3.5x speedup on dense models like Llama2-70B, its best performance on the Mixtral 8x7B MoE model is a modest 1.5x. The main challenges are at both algorithm and system levels: the low acceptance rates and slow generation of draft tokens; the high verification cost using the MoE model.

Limitations of existing approaches arise from a fundamental disconnect: system-level optimizations have overlooked the potential of integrating algorithmic innovations like speculative decoding, thereby missing optimization opportunities. Conversely, existing speculative decoding methods are purely algorithmic and fail to account for the system-level bottlenecks of MoE inference, especially on resource-constrained devices. Our key insight is that *not all activated experts are needed for generating a token*; a small subset of the most influential experts can often produce a high-quality draft token. This observation, supported by our empirical findings that model accuracy degrades gracefully as the number of active

*Corresponding author



WWW '26, Dubai, United Arab Emirates

ACM ISBN 979-8-4007-2307-0/2026/04
https://doi.org/10.1145/3774904.3792218

experts is reduced, reveals an opportunity for a more efficient inference paradigm. Leveraging this insight, we introduce SS-MoE, an algorithm-system co-design that optimizes MoE inference through the self-speculative decoding to achieve both accelerate and accurate. Our framework is built upon three core innovations:

- Efficient Prefetching for Self-Drafting and Verification: We employ a "self-drafting" mechanism where draft tokens are generated by the MoE model utilizing only a small, high-priority subset of the routed experts. To support the self-drafting and verification stages, we define a novel metric, $GoodPrefetch$, to guide training-free prefetching strategies (top-k/r experts) and consistently improve efficiency.
- GPU as the Expert Cache: To minimize I/O overhead, we treat the limited GPU memory as an LRU cache for partial experts, while all experts are stored in the CPU memory. This design eliminates the costly movement of data from the GPU back to the CPU, thereby reducing latency. Another advantage is that in-cache experts for draft tokens are often routed in the subsequent verification stage, thus reducing the total movement of experts.
- Confidence-Based Adaptive Acceptance: To address the challenge of multiple verification tokens of vanilla speculative decoding, we introduce a confidence-based policy. This allows confident draft tokens to be accepted while unconfident tokens are further verified, reducing the verification overhead required and boosting end-to-end speed while preserving model accuracy.

To the best of our knowledge, SS-MoE is the first work to adopt speculative decoding for accelerating on-device MoE inference. By co-designing the self-speculative decoding algorithm and memory system, our framework achieves both high performance and high accuracy. Our evaluations on a consumer-grade RTX 4090 GPU show that SS-MoE, running Moonlight-16B-A3B-Instruct, achieves a state-of-the-art inference speed of 23.15 tokens/s with nearly lossless accuracy. This represents a 3.72x speedup over leading baselines and even surpasses 4-bit quantization methods, which are completely accommodated in memory, by 1.15x.

2 Background and Motivation

2.1 Speculative Decoding

Speculative decoding (SD) is a technique designed to accelerate the auto-regressive generation process of LLMs. The process operates in two phases: a drafting phase by a draft model and a verification phase by the target model. In the drafting phase, a smaller, faster draft model generates a sequence of candidate tokens based on the current prompt. Subsequently, during the verification phase, the target model performs a single, parallel forward pass on the entire draft sequence and computes the probability distributions. The draft tokens are then validated sequentially using speculative sampling [8]. If a draft token is accepted, the process continues to the next token. If a token is rejected, a new token is sampled from a modified probability distribution, and all subsequent draft tokens are discarded. Finally, the sequence of accepted and newly sampled tokens is appended to the prompt, and the entire cycle repeats until generation is complete. In summary, SD transforms auto-regressive token generation to generating multiple tokens at a single forward pass, effectively parallelizing the process and leading to better utilization of computational resources.

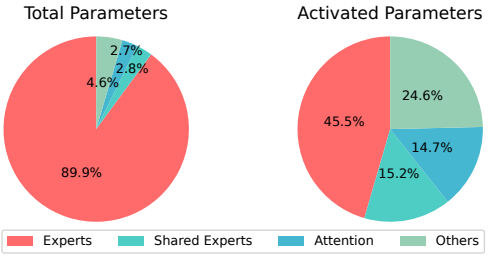


Figure 1: MoE model parameters composition in total and activated parameters during decoding.

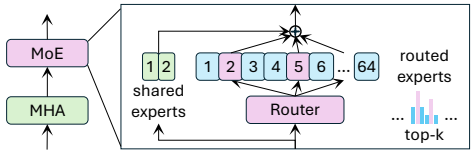


Figure 2: Sparse MoE architecture.

2.2 MoE Architecture

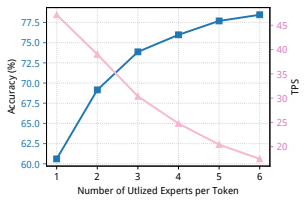
MoE architectures have been widely adopted by state-of-the-art LLMs, including Mixtral [11], DeepSeek [1], Qwen [12], and Gemini [13]. An MoE model is primarily composed of several key components: attention blocks, MoE blocks, a language model head (LM Head), a token embedding table, and standard layers such as Normalization and Dropout. As illustrated by the DeepSeek-V2-Lite-Chat (DeepSeek model) in Figure 1, the expert weights constitute the vast majority of the model's parameters, accounting for approximately 90% of the total. However, sparse MoE only activates a small portion (1/8) of experts during inference. For every decoding token, the primary computational load originates from the MoE and attention blocks, while the dominant latency is due to memory IO overhead for loading experts [3]. This inherent sparsity presents a significant opportunity for optimization. Modern sparse MoE employs a fine-grained architecture with shared and routed experts with a small hidden size, as illustrated in Figure 2.

The gating function, denoted as $G(x)$, is responsible for assigning routing weights to the experts for each input token x. We consider two primary types of gating functions, denoted as $G_1(x)$ and $G_2(x)$, which are employed in architectures such as DeepSeek-V2 and DeepSeek-V3, respectively. Let N be the total number of routed experts and k be the number of experts routed per token. The gating process first computes logits using weight matrix W_g. The TopK function then selects the k experts with the highest logits. Given M shared experts $S_0, S_1, \ldots, S_{M-1}$ and N routed experts $E_0, E_1, \ldots, E_{N-1}$, the final output of the MoE layer is the sum of the outputs from the shared experts and the weighted sum of the outputs from the activated routed experts. The output is as follows:

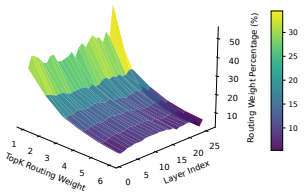
$$G_1(x) = \text{TopK}(\text{Softmax}(x \cdot W_g)) \tag{1}$$

$$G_2(x) = \text{Normalization}(\text{TopK}(\text{Sigmoid}(x \cdot W_g))) \tag{2}$$

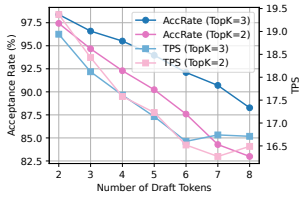
$$y(x) = \sum_{i=0}^{N-1} G(x)i \cdot E_i(x) + \sum_{j=0}^{M-1} S_j(x) \tag{3}$$



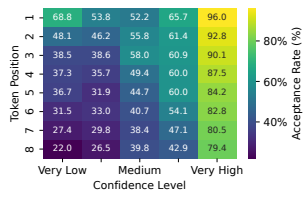
(a) ARC-Easy accuracy and decode speed using fewer experts.



(b) Routing weight distribution for each token across layers.



(c) Acceptance rate and end-to-end TPS of vanilla self-SD.



(d) Draft token acceptance rates under different confidence.

Figure 3: Performance analysis of the MoE model, including accuracy (a), routing weights (b), and self-SD performance (c) and acceptance rates (d).

Here, $G(x)_i$ represents the weight assigned to the i-th expert for the input x. Note that for any given token, only k of these weights are non-zero. For instance, the DeepSeek model has 2 shared experts and 64 sparsely activated experts, of which 6 experts are routed for one token.

2.3 Insight: Sparsity Exists In Routed Experts

> Insight 1. *A subset of the routed experts can achieve accuracy comparable to the full model.*

Our approach is motivated by a key observation: as illustrated in Figure 3a, a small subset of routed experts of an MoE model (e.g., the DeepSeek model) can still preserve comparable accuracy. Employing just the top-3/6 experts yields nearly 94% of the upbound accuracy. Even when only 1 routed expert is used, the degraded accuracy of 60.61% still has most (77.3%) of the upbound accuracy. This accuracy phenomenon is rooted in the highly skewed distribution of the routing weights, as shown in Figure 3b. The top-1 routing weights occupy around 40%, and the sum of the top 2 consistently occupies over 60% of the cumulative routing weights. Moreover, the decoding TPS of fewer experts is much faster than that of full experts, as shown in Figure 3a. Specifically, using a top-1 expert is 2.67x faster than using top-6 experts under the GPU with 24GB memory serving the DeepSeek model. In summary, using fewer routed experts is not only mostly accurate but also more efficient.

> Insight 2. *Confident self-drafted tokens are highly likely to be accepted by the target model.*

To investigate the speculative decoding method based on self-drafting, we conduct an experiment using the DeepSeek model and *Math* dataset, results shown in Figure 3c. As the draft length

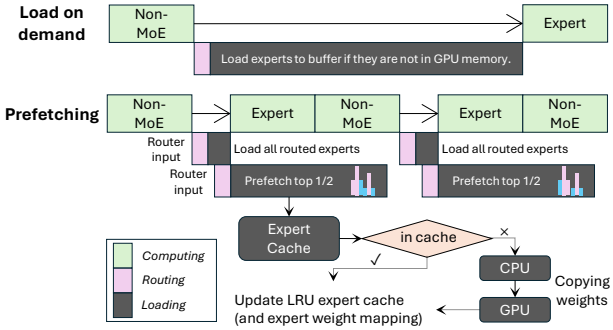


Figure 4: Expert cache and overlapped prefetching.

increases, both the average acceptance rate and decode speed decrease, which results from the verification overhead linearly increasing with the draft token counts. However, we find that the draft tokens have an average acceptance probability of over 90%. Specifically, the two draft tokens generated by top-2 experts have an average acceptance rate of around 97.5%. Moreover, their acceptance rates have a strong positive correlation with the draft confidence at every token position, as illustrated in Figure 3d, such as the 96.0% acceptance rate for the first token with high confidence and 68.8% for that with low confidence. This phenomenon inspires us that verification for confident tokens can be skipped to speedup decoding without harming the model accuracy.

Opportunity In summary, a top-k/r configuration maintains high model accuracy while enabling efficient system-level execution. This MoE property presents a powerful optimization opportunity. It suggests that a subset of routed experts can serve as a highly accurate and efficient draft model. This insight forms the cornerstone of our approach: we can leverage this smaller, faster configuration to rapidly generate accurate draft tokens, which can then be efficiently verified through a confidence policy.

3 Method

Figure 4 shows the fundamental memory design for the decoding process. Furthermore, Algorithm 1 presents a detailed confidence policy for general self-SD. The overview of `SS-MoE` is illustrated in Figure 6, which is built on the memory design and contains the confidence policy, illustrated using two draft tokens.

3.1 Expert Cache and Prefetching Design

As shown in the Figure 4, "expert cache" refers to the CPU storing all experts, while the GPU, after loading the model's non-expert components, allocates an expert memory pool based on the remaining available memory. Second, "prefetching" involves using the input of the current layer's MoE block to feed into the next layer's router. This strategy effectively hides the expert loading latency within the computation time of the current layer's MoE and the subsequent layer's attention block. A key feature of our method is that the expert memory pool only requires GPU weight updates, which avoids the overhead of copying weights back to the CPU. Furthermore, we employ a least recently used (LRU) cache, where the most recently used experts are retained in the cache for efficient subsequent access. In both stages (a) and (b), illustrated in Figure 6, the expert and prefetching design plays a crucial role.

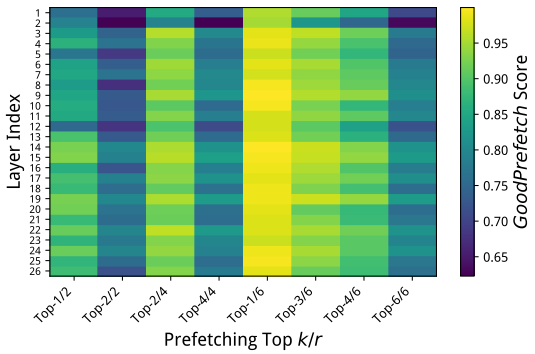


Figure 5: *GoodPrefetch* **score across layers and strategies.**

GoodPrefetch: Prefetching is a commonly adopted method to hide the loading latency of experts to be routed in the next layer within the computation of the current layer. It uses input from the last layer MoE to the router of the current layer to predict. The feasibility of this method is based on the highly similar hidden states of the continuous layer. However, the accuracy of the prediction cannot be guaranteed. Moreover, when loading latency cannot be fully hidden, the end-to-end acceleration is not decided by the accuracy of prefetching but by the ratio of truly utilized experts during prefetching.

$$GoodPrefetch = N_u/N_p \tag{4}$$

N_p denotes the number of prefetching experts, and N_u refers to the number of utilized prefetching experts. We use the notation top-k/r to denote the k highest-weighted experts out of the r experts routed for one token. For example, when the next layer requires 6 experts, prefetching the top 6 experts, including 1 truly routed expert, will cause the unwanted latency of 5 expert prefetching. In this case, the end-to-end acceleration is significantly impaired by incorrect prefetching. In contrast, prefetching top-1/6 truly routed experts can achieve no waste of loading overhead, thus accelerating end-to-end inference. For instance, prefetching the top-1/6 expert for the DeepSeek model results in a 98.30% *GoodPrefetch* score.

Based on this observation, we customize our method by prefetching **top-k/r** experts, where $k < r$, guaranteeing a high *GoodPrefetch* score and acceleration. This means that when the next layer's MoE router requires r experts, we only prefetch the top k experts. Empirically, we find that a prefetching ratio of **top-1/2** is quite effective (shown in Figure 5). We gather predictive and actually routed experts using DeepSeek model and Wikitext dataset. For instance, in stage (a), a top-1/2 prefetching strategy achieves 86.06% *GoodPrefetch* score for end-to-end acceleration. Similarly, in stage (c), prefetching top-3/6 for DeepSeek results in 93.40% *GoodPrefetch* score and also provides end-to-end speedup. In contrast, prefetching top-2/2, top-4/4, or top-6/6 causes an at most 77.52% *GoodPrefetch* score and slows down the overall speed.

Finally, our method capitalizes on the properties of the LRU cache and the *expert locality* of the consecutive draft & verify stages. Experts activated by tokens during the draft stage are frequently reactivated during the verify stage. Consequently, the cache can retain the experts loaded during the draft stage for the subsequent verification stage. This significantly reduces the overhead of expert memory updates at no additional cost.

3.2 Confidence Policy for Self-SD

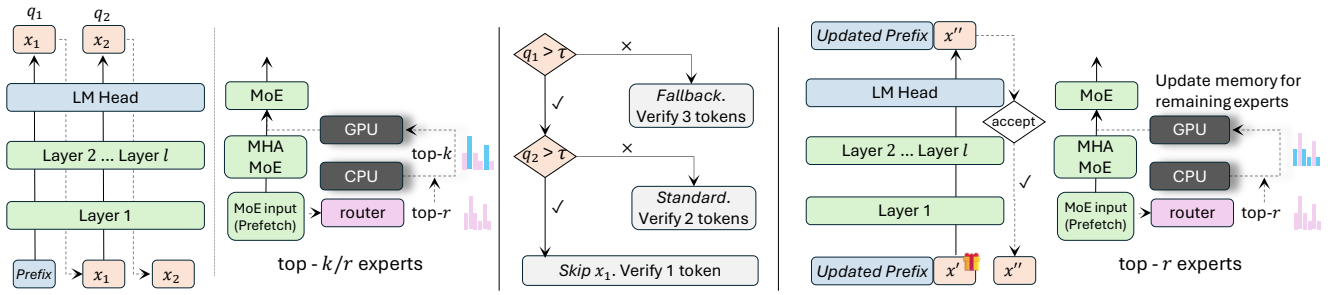
Algorithm 1: SS-MoE

Input: model M, prefix $\mathbf{x} = [x_1, x_2, \ldots, x_n]$, KV cache **KV**, max generation tokens L, draft token budget γ, draft experts k, confidence threshold τ

1 **Initialize:** prefill prefix $\mathbf{x}$ for target model
2 **while** $n - \mathbf{x}.\texttt{length} < L$ **do**
3 // Self-drafting forward using top-k experts:
$\mathbf{q}_{[\gamma]}, \hat{\mathbf{x}}_{[\gamma]} \leftarrow M(k,\ \mathbf{x}_{\leq n},\ \gamma)$;
4 // Confidence policy:
$\lambda \leftarrow \gamma$ // Initialize as all tokens are confident
5 **for** $j = 1$ **to** γ **do**
6 **if** $q_j < \tau$ **then**
7 $\lambda \leftarrow j - 1$ // Record position before first low-confidence token
8 **break**
9 **end**
10 **end**
11 $\lambda \leftarrow (\lambda - 1)$ // A conservative step back
12 // Rollback KV cache with $\gamma - \lambda$ tokens
$\mathbf{KV} \leftarrow \mathbf{KV}_{[1:\mathbf{KV}.\texttt{length}-\gamma+\lambda]}$
13 // Target model forward using full experts:
$\mathbf{p}_{\leq n} \leftarrow M(\mathbf{x}_{\leq n})$;
14 // Verification (Standard SD Algorithm [3])
$n, \mathbf{x} \leftarrow \texttt{Verify}(n, \mathbf{x}, \mathbf{q}_{[:\gamma-\lambda]}, \hat{\mathbf{x}}_{[:\gamma-\lambda]}, \mathbf{p}_{\leq n})$
15 **end**

The core idea of our confidence policy is built upon the insight that self-drafting can generate high-quality tokens, and the confidence level serves as a reliable proxy for token quality and acceptance rates. However, solely relying on the draft model to generate all tokens leads to a substantial drop in accuracy; for example, on the GSM8K benchmark, the DeepSeek model's accuracy falls from 46.34% with top-6 experts to 34.76% with top-2 experts. Conversely, if every drafted token is subjected to verification, the acceleration from speculative decoding becomes minimal or even negative, as our subsequent experiments will demonstrate in Figure 9. To resolve this dilemma, we have modified the speculative decoding algorithm to strike a balance between speed and accuracy.

λ is the number of tokens to be skipped during verification. $\mathbf{q}_{[\gamma]}$ denotes the confidence values (i.e., the probabilities of the sampled tokens) of γ draft tokens. Line 11 in our Algorithm 1 is designed to prevent the verification skipping of all confident draft tokens. We posit that the confidence of the first draft token alone is insufficient for it to be accepted without verification. Instead, the verification of the first draft token can only be skipped when the second consecutive draft token is also confident. Specifically, we limit the draft budget to 2 tokens, which simplifies the process into the conditional logic depicted in (b) in fig. 6. Based on the confidence levels of these 2 draft tokens, three outcomes are possible: (1). Fallback: Recompute one token. (2). Standard: Proceed without skipping verification for any token. (3). Skip First Token: Skip the verification of the first draft token. Our method utilizes scenarios (1) and (2) to maintain or even slightly improve accuracy, while leveraging scenario (3) to



(a) Draft with fewer experts (b) Confidence policy (c) Verify with full experts

Figure 6: Overview of SS-MoE, featuring a self-drafting stage and an additional confidence policy.

achieve nearly lossless acceleration. In a more general case where the draft budget exceeds two, our method consistently ensures that the final draft token is not skipped. This guarantees that at least the last token is always verified by the target model, preventing the process from devolving into the draft model generating all tokens independently. This safeguard is crucial for preserving the overall inference accuracy.

In summary, our algorithm-system co-design creates a synergistic framework that optimizes both the self-drafting and verification stages. At the system level, our decoding process overlaps computation with the asynchronous updating in the LRU experts cache. This cache design provides an immediate benefit by making draft experts readily available for the verification stage at no additional cost, which significantly reduces the number of experts that need to be loaded and accelerates the wall-clock time. The algorithmic layer (our confidence policy) complements this system-level optimization. By selectively determining the number of tokens that require verification, the policy intelligently reduces the workload for the verification stage. This minimizes the number of tokens to verify, further decreasing the demand for loading new experts. Together, these components ensure high accuracy while maximizing efficiency, effectively resolving the trade-off between speed and accuracy in on-device MoE model inference.

4 Experiment

4.1 Setup

MoE Models and Hardware. We use DeepSeek and Moonlight in this paper to refer to DeepSeek-V2-Lite-Chat[15] and Moonlight-16B-A3B-Instruct[16]. Architecturally, both models feature 64 routed experts, 2 shared experts, and an expert intermediate dimension of 1408. For each token, the router selects the top 6 experts. The primary distinction lies in their routing mechanism: DeepSeek employs a standard softmax function for calculating routing weights, whereas Moonlight utilizes a sigmoid function coupled with a scaling factor of 2.446 and a normalized top-k probability. The DeepSeek model is the default, with no additional comments. We conduct all experiments on a workstation equipped with an Intel Core i9-13900KF CPU, 24 cores, 32 threads, 64 GB of RAM, and an NVIDIA GeForce RTX 4090 GPU with 24 GB of memory, connected via a PCIe 4.0 x16 interface.

Datasets. Our evaluation builds on the open-source SD framework (SpecBench [17]) across a diverse set of tasks. We utilize benchmarks including MT-bench [18] for multi-turn dialogue, WMT14 DE-EN [19] for machine translation, CNN/Daily Mail [20] for summarization, Natural Questions [21] for question answering, GSM8k [22] for mathematical reasoning, and HumanEval [23] for code generation. For retrieval-augmented generation, we utilize Natural Questions, with five concatenated Wikipedia documents from the DPR dataset [24]. Additionally, TruthfulQA [25] (which requires generating answers to questions), GSM8k, and HumanEval are selected for evaluating decoding accuracy.

Baselines. We compare our approach against both system-level and algorithm-level methods for autoregressive generation. The system-level baselines include:

Demand [3], an on-demand expert loading strategy;

Fiddler [26], which offloads expert computation to the CPU to avoid weight transmission at the cost of hidden states transmission;

Daop [6], which further attempts to hide latency by pre-calculating experts on the CPU at the cost of decoding accuracy.

As a performance upbound, we also include popular 4-bit model quantization methods, *AWQ* [27] and *bitsandbytes*, which enable model inference to be completely accommodated in GPU memory.

Regarding the implementation of the baseline, it is important to note that while the original Fiddler and Daop methods incorporate CPU-GPU load balancing, their source code involves iterating through 2^N expert combinations, which is a specific design for models with a small number of experts, such as Mixtral-8x7B (where $N = 8$). This approach becomes computationally intractable for models with a large number of experts, like the 64-expert model used in our work. Therefore, our implementation of these baselines focuses on their core offloading mechanism, excluding this specific load-balancing feature.

To evaluate the individual contributions of our components, we also analyze several ablations of our own method. These include: Memo, which isolates the expert cache design detailed in Section 3.1; and $\lambda = 0$, which employs self-speculative decoding built upon the Memo but without the confidence policy from Section 3.2 (i.e., all draft tokens are verified). For all generative tasks, we set the generation parameters according to the official configuration provided on the Hugging Face page to ensure a reproducible comparison.

Table 1: Decoding performance across different models and tasks. We report decoding *TPS* and speedup ratio (φ) relative to load-on-demand autoregressive generation. The best results are highlighted in bold.

	Method		MT-Bench		Trans		Sum		QA		Math		RAG		Code		Avg.	
			TPS	φ	*TPS*	φ	*TPS*	φ	*TPS*	φ	*TPS*	φ	*TPS*	φ	*TPS*	φ	*TPS*	φ
DeepSeek	Demand		6.43	1.00	6.96	1.00	6.79	1.00	7.17	1.00	7.03	1.00	7.15	1.00	7.03	1.00	6.94	1.00
	Fiddler		6.05	0.94	6.06	0.87	6.06	0.89	6.20	0.86	6.04	0.86	6.27	0.88	6.04	0.86	6.10	0.88
	Daop		5.90	0.92	6.41	0.92	6.20	0.91	6.68	0.93	6.59	0.94	6.68	0.93	6.53	0.93	6.43	0.93
	4-bit		15.74	2.45	15.95	2.29	15.76	2.32	16.60	2.32	16.04	2.28	16.42	2.30	16.01	2.28	*16.07*	*2.32*
	Memo		15.28	2.38	15.91	2.29	14.58	2.08	15.41	2.15	14.41	2.05	15.06	2.11	13.93	1.98	**14.94**	**2.15**
	T=0	$\lambda = 0$	17.49	2.72	17.52	2.52	16.00	2.36	14.22	1.98	15.78	2.24	14.89	2.08	16.08	2.29	16.00	2.31
		SS-MoE	18.73	2.91	19.72	2.83	18.20	2.68	16.58	2.31	18.24	2.59	16.40	2.29	18.75	2.67	**18.09**	**2.61**
	T>0	$\lambda = 0$	17.35	2.70	17.79	2.56	16.73	2.46	17.12	2.39	16.33	2.32	16.61	2.32	15.65	2.23	16.80	2.42
		SS-MoE	18.75	2.92	18.73	2.69	17.86	2.63	18.57	2.59	18.60	2.65	17.70	2.48	18.14	2.58	**18.34**	**2.65**
Moonlight	Demand		5.91	1.00	6.35	1.00	5.85	1.00	6.25	1.00	6.30	1.00	6.32	1.00	6.46	1.00	6.21	1.00
	Fiddler		5.85	0.98	5.90	0.93	5.83	1.00	6.00	0.96	5.78	0.92	5.92	0.94	5.91	0.92	5.88	0.95
	Daop		5.47	0.93	5.66	0.89	5.16	0.88	5.89	0.94	5.77	0.92	6.17	0.98	5.84	0.90	5.71	0.92
	4-bit		19.59	3.31	20.15	3.17	19.80	3.38	20.77	3.32	20.25	3.21	20.27	3.21	20.15	3.12	*20.14*	*3.24*
	Memo		16.38	2.77	17.94	2.83	14.59	2.49	17.15	2.74	18.21	2.89	17.26	2.73	20.32	3.15	**17.41**	**2.80**
	T=0	$\lambda = 0$	17.11	2.90	19.40	3.06	16.58	2.83	18.79	3.01	21.53	3.42	18.91	2.99	25.12	3.89	19.63	3.16
		SS-MoE	19.70	3.33	20.83	3.28	18.37	3.14	20.40	3.26	24.03	3.81	20.14	3.19	26.65	4.13	**21.45**	**3.45**
	T>0	$\lambda = 0$	17.37	2.94	22.55	3.55	18.09	3.09	21.07	3.37	24.33	3.86	20.75	3.28	25.44	3.94	21.37	3.43
		SS-MoE	19.82	3.35	24.90	3.92	19.79	3.38	23.17	3.71	25.63	4.07	22.16	3.51	26.58	4.11	**23.15**	**3.72**

4.2 Speedup Result

As shown in the Table 1, our method achieves state-of-the-art performance, consistently outperforming all baselines across all datasets and models. We present our evaluation by decoupling baselines and methods into system-level and algorithm-level, and analyzing the effectiveness of SD-based algorithms under different decoding temperatures (T), where $T = 0$ denotes greedy decoding and $T > 0$ indicates sampling.

System-level superiority: At the system level, our Memo framework demonstrates significant superiority, proving highly effective across different models. For the DeepSeek model, it achieves a throughput of 14.94 TPS, while for the Moonlight model, it reaches 17.41 TPS. These speeds closely approach those of their respective 4-bit quantized versions, which run at 16.07 TPS and 20.14 TPS, which are completely accommodated in GPU memory. This highlights the advantage of our approach: delivering performance comparable to quantization without incurring the model accuracy loss. These results confirm that our system-level method provides a robust and efficient solution that is effective for various MoE models.

Inefficiency of CPU computation offloading: We observe that methods reliant on CPU offloading, i.e., Fiddler and Daop, perform poorly and fail to surpass the Demand baseline. For instance, on the Deepseek model, Fiddler's throughput was merely 0.88x that of the Demand baseline. We attribute this inefficiency to two factors: insufficient CPU compute power and the relatively low overhead of loading experts in fine-grained sparse MoE models (e.g, DeepSeek's expert dimension is only 1408). Consequently, offloading computation to the CPU introduces latency that hinders, rather than accelerates, overall inference speed. Improving such offloading methods for various models is not a trivial task. It would necessitate complex and highly detailed load-balancing strategies, considering the heterogeneity of CPUs and GPUs, as well as the large number (e.g., 64 or more) of experts required for fine-grained MoE architectures.

System-algorithm co-design synergy: Building on the Memo system, our algorithm-level SD methods yield further acceleration, boosting throughput from 14.94 to 16.00 TPS (with a fixed draft budget $\gamma = 2$). This result validates the effectiveness of our system-algorithm co-design, which capitalizes on cached expert locality between draft & verify. We further analyze the expert locality phenomenon in Section 4.4.1. Furthermore, performance improves significantly with sampling ($T > 0$), as it relaxes draft token verification, allowing less probable tokens to be accepted. This effect lifts Moonlight model throughput from 19.63 to 21.37 TPS and is even more pronounced when confidence policy is enabled (adaptive λ), increasing from 21.45 to 23.15 TPS. SS-MoE demonstrates remarkable acceleration, attaining maximum velocity and an additional 0.29x speedup with token distribution shifts, while maintaining comparable accuracy. In subsequent analyses in Section 4.4.3, this advantage becomes even more pronounced with smaller GPU memory budgets, resulting in greater speedup ratios. Ultimately, SS-MoE provides lossless acceleration, surpassing the speed of system-level baselines and even the quantization method.

4.3 Accuracy Result

Table 2 illustrates the impact of SS-MoE on decoding accuracy, evaluated on three representative datasets to assess both general and reasoning capabilities. In the table, "Up" refers to the baseline accuracy of the original model, while "Down" indicates the accuracy from a degraded setting using only the top-2 experts for decoding. For our method, the confidence threshold τ is fixed at 0.95. Overall, our method effectively maintains the accuracy of the model, even with an average accuracy increase of 0.02% for the DeepSeek model, and a negligible decrease of only 0.23% for the Moonlight model.

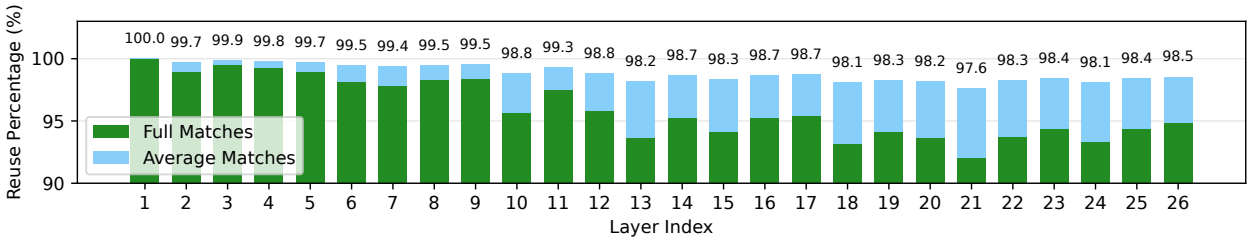


Figure 7: Expert locality indicated by layer-wise reuse between self-drafting and verification stages.

Table 2: Decoding accuracy (%) using SS-MoE in comparison with the upbound and downbound across generative tasks.

Model	Method	HumanEval	TruthfulQA		GSM8K	Avg.	Δ
		Pass@1	R1[a]	R2[a]	ExactMatch		
DeepSeek	Up	46.34	45.29	37.33	46.50	44.72	-
	Down	34.76	42.35	35.01	35.30	36.25	-8.47
	Ours	47.56	44.68	35.25	46.70	44.74	+0.02
Moonlight	Up	57.92	38.55	30.72	42.90	45.15	-
	Down	35.36	39.65	32.31	38.00	36.45	-8.70
	Ours	53.66	39.77	31.82	45.30	44.92	-0.23

[a]R1 and R2 denote the Rouge-1 and Rouge-2 scores, respectively.

This result demonstrates that the tokens skipped by our confidence policy have a minimal effect on task performance.

Notably, the impact of our method varies across different datasets and models. On the TruthfulQA dataset, which measures the recall capability of LLMs by assessing the similarity between generated and reference texts, our method yields mixed results. It slightly degrades the R1 score for the DeepSeek model, whereas for the Moonlight model, R1 surprisingly increases from 38.55% to 39.77%. A further complication is that the Down setting for the Moonlight model scores higher on TruthfulQA than its Up counterpart. This suggests that for certain recall-oriented tasks, model confidence is not always a reliable indicator of correctness. In contrast to this nuanced effect, our method delivers a surprising and consistent improvement in reasoning performance. On the reasoning-intensive benchmarks, HumanEval and GSM8K, SS-MoE not only preserves but slightly improves accuracy across the board. For instance, the DeepSeek model's accuracy on GSM8K increases from 46.34% to 47.56%, a notable improvement of 1.22%. Similarly, the Moonlight model's accuracy on the same benchmark rises from 42.9% to 45.3%. We hypothesize that this stems from the model generating more accurate tokens when its confidence is high, which in turn guides the reasoning process toward a correct solution. This phenomenon is consistent with recent observations [28].

4.4 Analysis and Ablation Study

4.4.1 Expert Locality. As illustrated in Figure 7, we analyze the expert reuse percentages using the DeepSeek model on the Math dataset to investigate the phenomenon of expert locality. The figure shows the proportion of experts used during the self-drafting phase that are subsequently reused during the verification phase. Our findings reveal a remarkable degree of overlap: the average proportion of reused experts (Average Matches) consistently falls between 97.6% and 100%. Moreover, the percentage of the entire

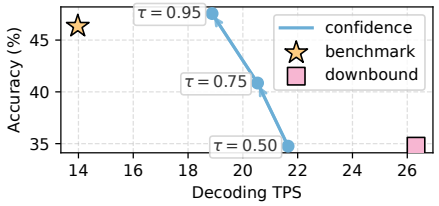


Figure 8: Impact of confidence τ on the speed and accuracy.

set of draft experts that is reused (Full Matches) is also substantial, remaining above 92.07%. This provides strong empirical evidence for the existence of significant expert locality.

We also observed that this locality is layer-dependent. The reuse percentage is highest (100%) in the first layer and gradually decreases with network depth, reaching its minimum at the 21st layer. We hypothesize that this trend occurs because the hidden states generated during the self-drafting phase progressively diverge from the target model's distribution as they pass through deeper layers, consequently altering the set of routed experts. This insight may inspire future designs for self-SD methods; for instance, the degree of drift in routing weights between the draft and verification stages could potentially be used as an early-exit criterion to predict whether a token will be accepted.

4.4.2 Confidence Study. As illustrated in Figure 8, the confidence can significantly increase the accuracy to the benchmark while preserving speedup. The benchmark and downbound refer to the DeepSeek model accuracy using top-6 and top-2 experts on the HumanEval dataset. As the confidence threshold, τ, is increased, our confidence policy progressively improves accuracy. This enhancement is accompanied by a marginal decrease in speed, as fewer tokens are skipped during verification. This relationship is depicted in the figure as a nearly linear trade-off between accuracy gains and performance reduction. Notably, at $\tau = 0.95$, our method achieves an accuracy that slightly surpasses the benchmark, while still maintaining a considerable speedup of 1.35x. This result effectively resolves the typical speed-accuracy dilemma. We observed that increasing τ beyond 0.95 yields no further accuracy gains and only reduces inference speed (not explicitly plotted in the figure). Therefore, based on this empirical sweet spot, we set $\tau = 0.95$ as the default for all models and datasets to achieve comparable accuracy.

4.4.3 Utilization of GPU Memory Budget. As shown in Figure 9, our method consistently outperforms state-of-the-art baselines across varying GPU memory budgets. X axis refers to the proportion of remaining GPU memory after allocating to non-expert components. Firstly, the decoding speed of our method scales positively

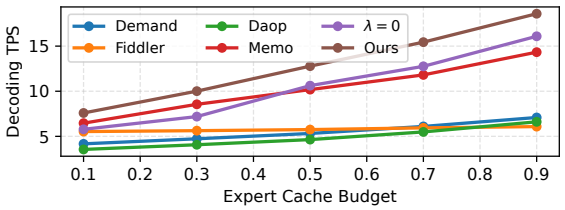


Figure 9: Decoding Performance under different memory budgets, using Deepseek model and Math dataset.

as the memory budget for the expert cache increases, demonstrating its ability to effectively leverage available resources for greater acceleration. Furthermore, our system-level optimization, Memo, consistently surpasses other system-level baselines. Among these baselines, we observe distinct behaviors: Fiddler performs well with small memory budgets but fails to scale, whereas Demand and Daop show clear performance gains as memory increases, ultimately overtaking Fiddler at budgets above 0.8. *Vanilla self-speculative decoding can slow down decoding under small memory budgets.* Without our confidence policy ($\lambda = 0$), self-SD can result in a slowdown, with TPS dropping below that of our system-level Memo baseline, because of increased overhead for loading experts during verification. For instance, at an expert cache budget of 0.3, 7.19 TPS for the $\lambda = 0$ configuration is lower than 8.55 TPS of Memo. In contrast, our proposed method (adaptive λ) consistently achieves SOTA performance across all tested memory budgets, outperforming both the Memo baseline and vanilla self-SD.

5 Related Work

5.1 Post-training MoE Quantization

Quantization is adopted for compressing LLM weights and activations to a smaller bit-width, thus significantly reducing memory overhead. SmoothQuant [29] reduces the quantization error of activation outliers through prior weight quantization accordingly. GPTQ [30] proposes importance-aware layer-wise and mixed-precision for model weights. AWQ [31] identifies the activation outliers and accordingly determines weights scales. The core idea of protecting critical weights is also important in MoE, for example, M-SMoE [32] leverages routing statistics to merge the experts with redundancy and compresses them into low-rank alternatives. Milo [33] tackles the accuracy loss of 4-bit quantization and proposes compensators with adaptive rank selection to recover the loss. In summary, quantization reduces the memory overhead but requires further methods to recover accuracy. Moreover, our method is orthogonal to quantization methods.

5.2 LLMs Speculative Decoding

Speculative decoding [8, 34] features with accelerating auto-regressive models without harming accuracy, by verifying several tokens parallely. The main challenge is low acceptance rate and computation latency of draft model. To increase the acceptance rate, Eagle [9] trains the draft model to generate features instead of tokens, and gains 2.7x speedup. Furthermore, Pearl [35] parallelizes the sequential dependence of draft-then-verify framework and achieves 1.50× speedup on SD. Medusa [36] utilizes multiple trained language model heads to generate draft tokens and 2.2× speedup. MoESD [37] utilizes SD for accelerating MoE in medium batch size and obtains up to 2.29x speedup by system optimization, however, leaves the single-batch challenges unresolved. In summary, SD for MoE is still under-explored due to increased verification overhead with the number of draft tokens.

5.3 MoE Serving in Datacenter

Deploying large-scale MoE models in datacenters necessitates a distributed setup. A common strategy is expert parallelism (EP), where experts are partitioned across different GPUs [39]. However, EP introduces substantial latency due to the frequent and high-volume all-to-all communication required to gather expert outputs. To address this challenge, existing research has primarily focused on techniques that hide or reduce communication overhead. One approach involves overlapping communication with computation. For instance, ScMoE [40] attempts to predict the experts needed for subsequent layers based on the current layer's input, thereby hiding all-to-all dispatch during the current layer's computation, achieving a 70% overlap efficiency. Others have focused on optimizing the communication itself. Chimera [41] develops a communication fusion mechanism that improves network bandwidth by 1.23x within combined parallelism strategies. Similarly, NetMoe [42] observes that tokens within the same sample often share routed experts. By leveraging this locality, it uses dynamic programming for sample placement to convert expensive inter-node communication into more efficient intra-node communication, yielding up to a 1.67x efficiency gain in MoE training. In summary, these data center optimizations focus on mitigating communication bottlenecks. The insights from our work can similarly inspire future research on applying speculative decoding to distributed MoE serving.

6 Conclusion

In this paper, we introduce `SS-MoE`, a novel framework designed to accelerate on-device MoE inference. To address the challenges of the high latency of on-demand expert loading under limited GPU memory, we developed a self-speculative decoding method. This approach leverages the inherent ability of MoE models to generate high-quality draft tokens rapidly using only a small subset of experts. We also incorporate a confidence policy to identify high-quality tokens and dynamically adjust the number of verification tokens. Our framework is a system-algorithm co-design, achieving significant end-to-end acceleration. At the system level, all experts are stored in CPU memory while a dynamic LRU cache on the GPU manages active experts. This enables the self-drafting and verification stages to leverage expert locality, resulting in a substantial reduction in total expert loading overhead. At the algorithm level, our confidence policy ensures comparable accuracy while enabling more aggressive acceleration. As a result, `SS-MoE` achieves up to a 3.72x decoding speedup over the best-performing baseline, demonstrating an effective solution for on-device MoE deployment.

7 Acknowledgments

This work is supported by the Hong Kong Research Grants Council, General Research Fund under Grant No. 15225023, and the National Natural Science Foundation of China under Grant No. 62302184.

References

[1] Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. 2024. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066* (2024).

[2] Wenchao Xu, Jinyu Chen, Peirong Zheng, Xiaoquan Yi, Tianyi Tian, Wenhui Zhu, Quan Wan, Haozhao Wang, Yunfeng Fan, Qinliang Su, and Xuemin Shen. 2025. Deploying Foundation Model Powered Agent Services: A Survey. *IEEE Communications Surveys and Tutorials* (2025), 1–1. doi:10.1109/COMST.2025.3580745

[3] Artyom Eliseev and Denis Mazur. 2023. Fast inference of mixture-of-experts language models with offloading. *arXiv preprint arXiv:2312.17238* (2023).

[4] Shuzhang Zhong, Ling Liang, Yuan Wang, Runsheng Wang, Ru Huang, and Meng Li. 2024. AdapMoE: Adaptive sensitivity-based expert gating and management for efficient moe inference. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*. 1–9.

[5] Xiaoniu Song, Zihang Zhong, Rong Chen, and Haibo Chen. 2024. Promoe: Fast moe-based llm serving using proactive caching. *arXiv preprint arXiv:2410.22134* (2024).

[6] Yujie Zhang, Shivam Aggarwal, and Tulika Mitra. 2025. DAOP: Data-Aware Offloading and Predictive Pre-Calculation for Efficient MoE Inference. In *2025 Design, Automation and Test in Europe Conference (DATE)*. IEEE, 1–7.

[7] Rui Kong, Yuanchun Li, Qingtian Feng, Weijun Wang, Xiaozhou Ye, Ye Ouyang, Linghe Kong, and Yunxin Liu. 2024. SwapMoE: Serving Off-the-shelf MoE-based Large Language Models with Tunable Memory Budget. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 6710–6720.

[8] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2022. Fast inference from transformers via speculative decoding, 2023. *URL https://arxiv. org/abs/2211.17192* (2022).

[9] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty. In *International Conference on Machine Learning*. PMLR, 28935–28948.

[10] Haozhao Wang, Haoran Xu, Yichen Li, Yuan Xu, Ruixuan Li, and Tianwei Zhang. [n. d.]. FedCDA: Federated Learning with Cross-rounds Divergence-aware Aggregation. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*.

[11] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[12] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388* (2025).

[13] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530* (2024).

[14] Haozhao Wang, Yabo Jia, Meng Zhang, Qinghao Hu, Hao Ren, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2024. FedDSE: Distribution-aware Sub-model Extraction for Federated Learning over Resource-constrained Devices. In *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024*, Tat-Seng Chua, Chong-Wah Ngo, Ravi Kumar, Hady W. Lauw, and Roy Ka-Wei Lee (Eds.). ACM, 2902–2913.

[15] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434* (2024).

[16] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, Yanru Chen, Huabin Zheng, Yibo Liu, Shaowei Liu, Bohong Yin, Weiran He, Han Zhu, Yuzhi Wang, Jianzhou Wang, Mengnan Dong, Zheng Zhang, Yongsheng Kang, Hao Zhang, Xinran Xu, Yutao Zhang, Yuxin Wu, Xinyu Zhou, and Zhilin Yang. 2025. Muon is Scalable for LLM Training. arXiv:2502.16982 [cs.LG] https://arxiv.org/abs/2502.16982

[17] Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. 2024. Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding. In *Findings of the Association for Computational Linguistics ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand and virtual meeting, 7655–7671. doi:10.18653/v1/2024.findings-acl.456

[18] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems* 36 (2023), 46595–46623.

[19] Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 Workshop on Statistical Machine Translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Christof Monz, Matt Post, and Lucia Specia (Eds.). Association for Computational Linguistics, Baltimore, Maryland, USA, 12–58. doi:10.3115/v1/W14-3302

[20] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. 2016. Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*. 280–290.

[21] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics* 7 (2019), 453–466.

[22] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168* (2021).

[23] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG] https://arxiv.org/abs/2107.03374

[24] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 6769–6781. doi:10.18653/V1/2020.EMNLP-MAIN.550

[25] Stephanie Lin, Jacob Hilton, and Owain Evans. 2021. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958* (2021).

[26] Keisuke Kamahori et al. 2024. Fiddler: CPU-GPU Orchestration for Fast Inference of Mixture-of-Experts Models. *arXiv preprint arXiv:2402.07033* (2024).

[27] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Proceedings of Machine Learning and Systems*, P. Gibbons, G. Pekhimenko, and C. De Sa (Eds.), Vol. 6. 87–100. https://proceedings.mlsys.org/paper_files/paper/2024/file/42a452cbafa9dd64e9ba4aa95cc1ef21-Paper-Conference.pdf

[28] Yudi Pawitan and Chris Holmes. 2024. Confidence in the reasoning of large language models. *arXiv preprint arXiv:2412.15296* (2024).

[29] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*. PMLR, 38087–38099.

[30] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. In *The Eleventh International Conference on Learning Representations*.

[31] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Proceedings of Machine Learning and Systems*, P. Gibbons, G. Pekhimenko, and C. De Sa (Eds.), Vol. 6. 87–100. https://proceedings.mlsys.org/paper_files/paper/2024/file/42a452cbafa9dd64e9ba4aa95cc1ef21-Paper-Conference.pdf

[32] Pingzhi Li, Zhenyu Zhang, Prateek Yadav, Yi-Lin Sung, Yu Cheng, Mohit Bansal, and Tianlong Chen. 2024. Merge, Then Compress: Demystify Efficient SMoE with Hints from Its Routing Policy. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=eFWG9Cy3WK

[33] Beichen Huang, Yueming Yuan, ZELEI SHAO, and Minjia Zhang. 2025. MiLo: Efficient Quantized MoE Inference with Mixture of Low-Rank Compensators. In *Eighth Conference on Machine Learning and Systems*. https://openreview.net/forum?id=NXVXiJmhe1

[34] Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*. PMLR, 19274–19286.

[35] Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, and Winston Hu. 2024. Parallel speculative decoding with adaptive draft length. *arXiv e-prints* (2024), arXiv–2408.

[36] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. In *International Conference on Machine Learning*. PMLR, 5209–5235.

[37] Zongle Huang, Lei Zhu, Zongyuan Zhan, Ting Hu, Weikai Mao, Xianzhi Yu, Yongpan Liu, and Tianyu Zhang. 2025. MoESD: Unveil Speculative Decoding's Potential for Accelerating Sparse MoE. *arXiv preprint arXiv:2505.19645* (2025).

[38] Peirong Zheng, Wenchao Xu, Haozhao Wang, Jinyu Chen, and Xuemin Shen. 2026. HALO: Semantic-Aware Distributed LLM Inference in Lossy Edge Network. arXiv:2601.11676 [cs.DC] https://arxiv.org/abs/2601.11676

[39] Xiaonan Nie, Pinxue Zhao, Xupeng Miao, Tong Zhao, and Bin Cui. 2022. HetuMoE: An efficient trillion-scale mixture-of-expert distributed training system. *arXiv preprint arXiv:2203.14685* (2022).

[40] Weilin Cai, Juyong Jiang, Le Qin, Junwei Cui, Sunghun Kim, and Jiayi Huang. 2024. Shortcut-connected expert parallelism for accelerating mixture-of-experts. *arXiv preprint arXiv:2404.05019* (2024).

[41] Le Qin, Junwei Cui, Weilin Cai, and Jiayi Huang. 2025. Chimera: Communication Fusion for Hybrid Parallelism in Large Language Models. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. Association for Computing Machinery, New York, NY, USA, 498–513. doi:10.1145/3695053.3731025

[42] Xinyi Liu, Yujie Wang, Fangcheng Fu, Xupeng Miao, Shenhan Zhu, Xiaonan Nie, and Bin Cui. 2025. Netmoe: Accelerating moe training through dynamic sample placement. In *The Thirteenth International Conference on Learning Representations*.